

# Задача переливання

*Water Jugs Problem*

Козін Андрій

НаУКМА • Логічне програмування • 2025

# Постановка задачі

## Дано:

- Набір посудин з відомими ємностями
- Початкові рівні води в кожній посудині
- Цільова кількість літрів

## Операції:

- `fill(I)` — наповнити посудину з джерела
- `empty(I)` — спорожнити посудину
- `pour(I,J)` — перелити з I в J

Приклад: [5л, 3л] → 4л

[5,0] → [2,3] → [2,0] →  
[0,2] → [5,2] → [4,3] ✓

# Алгоритм: пошук в ширину (BFS)

## Чому BFS, а не DFS?

- BFS гарантує оптимальний (найкоротший) розв'язок
- Досліджує стани рівень за рівнем
- Використовує чергу замість стеку

## Чому не CLP(FD)?

- Це задача пошуку послідовності дій
- CLP(FD) — для задач обмежень (судоку, розклади)

# Реалізація: Prolog

```
% bfs(+Queue, +Caps, +Goal, +Visited, -Solution) is semidet
bfs([[State, Path]|_], _, Goal, _, Path) :-
    member(Goal, State), !.
```

```
bfs([[State, Path]|Rest], Caps, Goal, Visited, Solution) :-
    findall([NewState, [Action|Path]],
        (move(State, Caps, NewState, Action),
         \+ member(NewState, Visited)),
        NewStates),
    append(Rest, NewStates, NewQueue),
    bfs(NewQueue, Caps, Goal, [State|Visited], Solution).
```

```
% move(+State, +Caps, -NewState, -Action) is nondet
move(State, Caps, NewState, pour(I, J)) :-
    nth1(I, State, LevelI), nth1(J, State, LevelJ), I \= J,
    LevelI > 0, nth1(J, Caps, CapJ), LevelJ < CapJ,
    Amount is min(LevelI, CapJ - LevelJ),
    NewLevelI is LevelI - Amount, NewLevelJ is LevelJ + Amount,
    set_nth(I, State, NewLevelI, Tmp), set_nth(J, Tmp, NewLevelJ, NewState).
```

# Порівняння: Haskell

```
solve :: Caps -> State -> Int -> Maybe [Action]
solve caps initial goal = bfs [(initial, [])] Set.empty
  where
    bfs [] _ = Nothing
    bfs ((state, path):rest) visited
      | goal `elem` state = Just (reverse path)
      | state `Set.member` visited = bfs rest visited
      | otherwise = bfs (rest ++ newStates) (Set.insert state visited)
    where
      newStates = [(s, a:path) | (a, s) <- moves caps state,
                                not (s `Set.member` visited)]
```

Аспект	Prolog	Haskell
Backtracking	Автоматичний	Явний (список)
Visited	Список	Data.Set
Типізація	Динамічна	Статична
Код	Коротший	Безпечніший

# Мультипризначеність предикатів

Предикат	Тип	Пояснення
solve/3	det	Один розв'язок (через cut)
bfs/5	semidet	Успіх або fail
move/4	nondet	Генерує варіанти через backtracking
set_nth/4	det	Один результат

## Індикатори параметрів:

- + вхідний (ground)
- - вихідний (unbound)
- ? може бути обидва

# Висновки

- Prolog — ідеальний для задач пошуку в просторі станів
- BFS гарантує оптимальний розв'язок
- Автоматичний backtracking спрощує генерацію станів
- Haskell потребує явного управління станом

Репозиторій: [github.com/kepeld/water-jugs-prolog](https://github.com/kepeld/water-jugs-prolog)

**Дякую за увагу!**